Lars Grunske, Janet Siegmund, Andreas Vogelsang (Hrsg.): SE 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 29

Identifying Domain-Based Cyclic Dependencies in Microservice APIs Using Source Code Detectors

Patric Genfer[1], Uwe Zdun[2]

Abstract: Isolation, autonomy, and loose coupling are critical success factors of microservice architectures, but unfortunately, systems tend to become strongly coupled over time and sometimes even exhibiting cyclic communication chains. These cycles can even manifest on a conceptual or domain level, making them hard to track for algorithms that rely solely on static analysis. Accordingly, previous attempts to detect cycles either focused on synchronous communication or had to collect additional runtime data, which can be costly and time-consuming. We suggest a novel approach for identifying and evaluating domain-based cyclic dependencies in microservice systems based on modular, reusable source code detectors. Based on the architecture model reconstructed by our detectors, we derived a set of architectural metrics for identifying and classifying domain-based cyclical dependencies. By conducting two case studies on open-source microservice architectures, we validated the feasibility and applicability of our approach.

Keywords: Microservice API; domain-based cyclic dependencies; metrics; source code detectors

1 Introduction

One of the main goals of microservices is to reduce the complexity of large monolithic applications by splitting them up into smaller, autonomously acting services [Th15]. In addition to being isolated from each other, lightweight inter-service communication is central in microservice architectures [Ne15]. These communications create dependencies between services and are often problematic when they form cycles, where a chain of service calls ends in the same service where it began [TL18]. Changing the communication flow from synchronous to asynchronous communication alone does not resolve these cyclic dependencies, but instead only shifts them to a different, more conceptual level [Wo17] where they now become part of the domain or business logic, making them even more difficult to track through static analysis. This work therefore presents a novel approach for identifying and evaluating both, synchronous and domain-based cyclic dependencies on microservice API operation level. For this, we reverse engineer a communication model from underlying microservice code artifacts by using lightweight source code detectors [Nt21] and based on this model, we define a set of architectural metrics for detecting and evaluating potential cyclic dependency structures.

[1] University of Vienna, Faculty of Computer Science, Research Group Software Architecture, Währinger Str. 29, 1090 Wien, Austria patric.genfer@univie.ac.at
[2] University of Vienna, Faculty of Computer Science, Research Group Software Architecture, Währinger Str. 29, 1090 Wien, Austria uwe.zdun@univie.ac.at

2 Architecture Reconstruction and Cycle Identification

To model the communication flow observable at the microservice API level, this paper uses a directed graph-based approach, similar to [Re18; ZNL17], but with a stronger focus on the inter-service communication. To reconstruct the microservice architecture from the underlying source code, we use a concept from our earlier research, called modular, reusable *source code detectors* [Nt21]. These lightweight source parsers scan the code according to predefined patterns and at the same time ignore any code artifacts unrelated to the communication model. While these detectors must still be adopted to identify technology-specific patterns, implementing and especially maintaining them requires less effort than comparable approaches like a complete AST reconstruction. Based on our formal communication model, we define a set of architectural metrics, both on service level but also on a more fine-grained API-level, that allow us to identify and assess cyclic dependencies within a microservice system.

3 Case Studies

We evaluated our approach by conducting two case studies with two different open-source microservice architectures, *Lakeside Mutual*[3] and *eShopOnContainers*[4], both taken from GitHub. We were able to identify domain-based cycles in both cases. Our case study has also shown that our approach is very well suited for agile development processes, as it requires only the underlying source code without gathering time-consuming runtime information, which makes it particularly interesting for continuous integration pipelines.

4 Conclusion

In this paper, we presented a novel approach for detecting technical and especially domain-based cyclic dependencies in microservice API architectures. Our approach confirms that the detection is possible by relying solely on static source code artifacts. While our source code detectors require some upfront implementation work, our case studies revealed that this effort is manageable and can also be reduced by reusing existing detectors where possible. The study results also show that by using our metrics, even inconspicuous domain-based cycles can be detected. The information gathered through our cycle analysis provides software experts with a solid foundation for making qualified decisions regarding a microservice system's architecture.

[3] https://github.com/Microservice-API-Patterns/LakesideMutual

[4] https://github.com/dotnet-architecture/eShopOnContainers

5 Data Availability

The source code and the data for the project are freely available. The data can be found under the following link: `https://swa.univie.ac.at/identifying-domain-based-cycles/`. This directory contains the generated communication models in png/svg format, together with the textual output of the cycle search and the calculated metrics for each case study. Source code for the detectors, the model generation and cycle search is available under `https://gitlab.swa.univie.ac.at/public-sources/microservice-cycles-public`.

References

[Ne15] Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Beijing Sebastopol, CA, 2015, ISBN: 978-1-4919-5035-7.

[Nt21] Ntentos, E.; Zdun, U.; Plakidas, K.; Genfer, P.; Geiger, S.; Meixner, S.; Hasselbring, W.: Detector-based component model abstraction for microservice-based systems. Computing/, pp. 1–31, 2021.

[Re18] Ren, Z.; Wang, W.; Wu, G.; Gao, C.; Chen, W.; Wei, J.; Huang, T.: Migrating Web Applications from Monolithic Structure to Microservices Architecture. In: Proceedings of the Tenth Asia-Pacific Symposium on Internetware. ACM, Beijing China, pp. 1–10, Sept. 2018, ISBN: 978-1-4503-6590-1.

[Th15] Thones, J.: Microservices. en, IEEE Software 32/1, pp. 116–116, Jan. 2015, ISSN: 0740-7459.

[TL18] Taibi, D.; Lenarduzzi, V.: On the definition of microservice bad smells. IEEE software 35/3, pp. 56–62, 2018.

[Wo17] Wolff, E.: Microservices: Flexible Software Architecture. Addison-Wesley, Boston, 2017, ISBN: 978-0-13-460241-7.

[ZNL17] Zdun, U.; Navarro, E.; Leymann, F.: Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns. In (Maximilien, M.; Vallecillo, A.; Wang, J.; Oriol, M., eds.): Service-Oriented Computing. Vol. 10601, Springer International Publishing, Cham, pp. 411–429, 2017, ISBN: 978-3-319-69034-6 978-3-319-69035-3.